Latest features of dpkg-dev

By Raphaël Hertzog
<hertzog@debian.org>

Mini-Debconf Paris
30-31 Oct 2010
Université Paris 7

What I'm going to cover

- New features of dpkg-dev in the last 2-3 years
- dpkg-dev: tools used to build debian packages
- In particular:
  - Support of symbols files by dpkg-shlibdeps, dpkg-gensymbols
  - Support of new source formats by dpkg-source
  - Supplementary options for dpkg-source
  - Cross distribution collaboration with dpkg-vendor
  - Custom compilation flags with dpkg-buildflags
  - Miscellaneous improvements to other tools

Support of symbols files

- Before: shlibs files: library used → dependency with minimal version
- Now: symbols files:
  - Library used → dependency template
  - Symbol used → minimal version for dependency
- dpkg-shlibdeps can thus generate more accurate dependencies with relaxed requirements in many cases
- dpkg-gensymbols needed to generate the symbols files

Example of shlibs/symbols files

Shlibs file:

```
libc 6 libc6 (>= 2.11)
```

Symbols file:

```
libc.so.6 libc6 #MINVER#
| libc6 (>> 2.11), libc6 (<< 2.12)
 abs@GLIBC_2.0 2.0
 accept4@GLIBC_2.10 2.10
 duplocale@GLIBC_2.3 2.3
 errno@GLIBC_PRIVATE 0 1
[...]
```

More symbols goodness

- Ensuring generated dependency is as strong as the corresponding build-dependency
- Example:

```
libgtk-x11-2.0.so.0 libgtk2.0-0 #MINVER#
* Build-Depends-Package: libgtk2.0-dev
 gtk_about_dialog_get_artists@Base 2.8.0
[...]
```

- If a package build-depends on libgtk2.0-dev (>> 2.12), the dependency on libgtk2.0-0 will be at least as strict (i.e. >> 2.12).

Adding symbols files in a source package

- You provide: debian/<package>.symbols
- dpkg-gensymbols process this file to generate the final symbols file
  - Add missing symbols
    - Dependency info from symbol patterns
    - Otherwise assumed to be new in this release
  - Fail if symbols/libraries got removed
  - Filter the set of symbols to include based on tags
- You must still keep the source file up to date (analyzing build logs is often enough)

Tags in debian/*.symbols files

- Dealing with arch-specific symbols
  ```
  (arch=i386 amd64)mysym@Base 1.0
  ```
- Marking private symbols which can disappear
  ```
  (optional)__internal_func@Base 1.0
  ```
- Matching symbols from the symbol version
  ```
  (symver)GLIBC_2.7 2.7
  ```
- Matching C++ symbols from demangled names
  ```
  (c++)"non-virtual thunk to
  NSB::ClassD::~ClassD()@Base" 1.0
  ```
- Matching symbols with a regular expression
  ```
  (regex)"^mystack_.*@Base$" 1.0
  ```
- Combining tags with “|”
  ```
  (regex|optional)^__internal 1.0
  ```

New formats for source packages

- Historical format: “1.0” (native + non-native)
- New formats accepted in the Debian archive
  - 3.0 (native) → like 1.0 native + default file ignore
  - 3.0 (quilt) → replacement for 1.0 non-native
- New experimental formats
  - 3.0 (git)
  - 3.0 (bzr)
- You tell dpkg-source what you want

```
mkdir -p debian/source
echo "3.0 (quilt)" > debian/source/format
```

“1.0” vs “3.0 (quilt)”

- 1.0
  - Packaging files:
    - in .diff.gz
  - Changes to upstream files:
    - Applied all at once by the .diff.gz during unpack
    - Applied by a custom patch management system (dpatch, quilt, …) during build
- 3.0 (quilt)
  - Packaging files:
    - in .debian.tar.*
  - Changes to upstream files:
    - Stored separately in debian/patches/
    - Applied during unpack

Features of 3.0 (quilt)

(that 1.0 doesn't have)

- Supports bzip2, xz, lzma
- Multiple upstream tarballs (.orig-<comp>.tar.*)
- Supports adding binary files (e.g. icons)
- Drops “debian” directory from upstream tarball
- Changes to upstream files → a dedicated patch in debian/patches/

debian

Basics of quilt patch system

- debian/patches/series: (ordered) list of patches to apply
- Adding a patch
  - quilt add name-of-patch
  - quilt edit upstream-file-to-modify
  - quilt refresh
- Updating fuzzy patches for a new upstream version
  - quilt pop -a
  - while quilt push; do quilt refresh; done

Passing options to dpkg-source

- Command-line options can be made sticky
- In the source package:
  debian/source/options
- In the VCS repository only:
  debian/source/local-options
- Syntax: command-line options without leading “--”, supplementary spaces allowed
- Example:

```
compression = "bzip2"
single-debian-patch
```

dpkg-source options for “3.0 (quilt)”

- --single-debian-patch: store changes in d/p/debian-changes instead of d/p/debian-changes-<version>
  - Useful only if you have a VCS-based workflow that doesn't allow you to generate a proper patch series, debian/source/patch-header should explain how changes can be reviewed
- --unapply-patches: after the build
  - When you build from a VCS and want the tree to be clean after build
- --create-empty-orig: bundle multiple software.

Other useful dpkg-source options

- --compression=bzip2
- --compression-level=9
- --extend-diff-ignore="(^|/)config.(sub|guess)$"
  - Ignore changes on some files when generating .diff.gz or d/p/debian-changes-<ver>
- --tar-ignore=unwanted-file
  - Do not include some files in native packages
- --abort-on-upstream-changes
  - Fail if upstream changes are not managed by an existing patch

Cross-distribution collaboration with dpkg-vendor

- Current vendor identified by /etc/dpkg/origins/default (→ debian/ubuntu)
- dpkg-vendor queries those files
- You can use dpkg-vendor in debian/rules to adjust the behavior during build

```
ifeq ($(shell dpkg-vendor --derives-from Ubuntu && echo yes),yes)
        SUBSTVARS = -Vdist:Depends="foo (>= 2)"
else
        SUBSTVARS = -Vdist:Depends="bar"
endif

%:
    dh $@

override_dh_gencontrol:
    dh_gencontrol -- $(SUBSTVARS)
```

dpkg-vendor interface

```
$ dpkg-vendor --is debian \
  && echo yes || echo no
yes
$ dpkg-vendor --derives-from ubuntu \
  && echo yes || echo no
no
$ dpkg-vendor --query Vendor-URL
http://www.debian.org/
$ man dpkg-vendor # for more info
```

Vendor-specific patch series

- Feature specific to 3.0 (quilt)
- dpkg-source uses a single quilt series file but tries in order:
  - debian/patches/<vendor>.series
  - debian/patches/series
- Debian should provide debian/source/series
- Derivatives can override it completely
- In general they should only extend it, and maybe drop Debian-specific branding patches

Customizing compilation flags with dpkg-buildflags

- Why do we want that?
  - Users who want to recompile for a 5% speed benefit (Gentoo like)
  - Changing default compilation flags at the distribution level (hardening, minimal CPU requirement changed, etc.)
  - Experimenting with new compilation flags
- How can we achieve it?
  - Packages need to inject flags returned by dpkg-buildflags in their build system

dpkg-buildflags

- Flags supported
  - CFLAGS
  - CPPFLAGS
  - CXXFLAGS //C++
  - FFLAGS //Fortran
  - LDFLAGS //Linker
  - dpkg-buildflags --list for more
- Values retrieved from
  - default value
  - /etc/dpkg/buildflags.conf
  - $HOME/.config/dpkg/buildflags.conf
  - $DEB_<flag>_SET
  - $DEB_<flag>_APPEND

```
dpkg-buildflags --get CFLAGS
dpkg-buildflags --export
```

dpkg-maintscript-helper

- Manages tedious operations usually done in maintainer scripts
  - Removing an obsolete conffile
  - Renaming a conffile
- Same snippet in all maintainer scripts.
- Example:

```
if dpkg-maintscript-helper supports \
   rm_conffile 2>/dev/null; then
        dpkg-maintscript-helper rm_conffile \
          /etc/foo/conf.d/bar 1.2-1 -- "$@"
fi
```

dpkg-shlibdeps

- Improved performance by caching data
- Generates lots of warnings that can be used to improve the binaries / the build system
  - --warnings=<mask> can be used to control the warnings displayed
- Enhanced support of cross compilation
- Mimicks ld.so well enough that it's rarely needed to give supplementary hints with LD_LIBRARY_PATH

dpkg-gencontrol

- Warns about unused substitution variables
- Supports multiple substvars files
- Simplify dependencies
  - a, a | b, a (>= 1) → a (>= 1)
- Fails on arch-specific dependencies if the packages is arch: all

dpkg-buildpackage

- -F → full build (source + binaries)
- -Rmyrules → run myrules instead of debian/rules
- --source-option=”--foo” → pass to dpkg-source
- --changes-option=”--foo” → pass to dpkg-genchanges
- -j → build in parallel, -jX → run X process in parallel

dpkg-mergechangelogs

- Problem: conflict on debian/changelog when merging the experimental branch of a package into the unstable branch
- Solution: custom merge script that understands how debian/changelog works
- Can be used with git (see manpage)
  - Register the merge driver in .git/config or ~/.gitconfig
  - Configure the repository to use that driver for debian/change in .git/info/attributes or .gitattributes

Questions ?

Grab the slides at:
http://raphaelhertzog.com/go/talk-2010-10-30

Credits & License

- Content by Raphaël Hertzog
  http://raphaelhertzog.com/go/talk-2010-10-30
  License: GPL-2+
- OpenOffice.org template by Raphaël Hertzog
  http://raphaelhertzog.com/go/ooo-template
  License: GPL-2+
- Background image by Alexis Younes “ayo”
  http://www.73lab.com
  License: GPL-2+